JAVA
Oppsummering for IS-102

Even Åby Larsen
even.larsen@uia.no

11. april 2013

Innhold

1 Innledning

Dette notatet er en oppsummering av det som er gjennomgått av Java så langt i semesteret. Denne versjonen oppsummerer til og med kapittel 4 i læreboka.

Det er meningen at notatet skal være et supplement til læreboken[1], ikke en erstatning. Forklaringene her er relativt kortfattede og forutsetter at læreboken er lest og forstått. Hvis du vil ha mer informasjon enn du finner i læreboken, finnes det flere "tutorials" på `http://docs.oracle.com/javase/tutorial/`[3].

2 Hva er et Java program

Et java program består av en eller flere klasser. Klassene fungerer som maler for å lage objekter. Objektene representerer "ting". Feltene til et objekt representerer egenskapene til "tingen". Metodene til objektet representerer handlinger som objektet kan utføre.

2.1 Kjøring av java-programmer

Vi kan kjøre et java program ved å laget et objekt, og kalle en av metodene til objektet. Det kan gjøres ved å bruke menyene i BlueJ.

Objektet utfører metoden ved å utføre setningene i metodekroppen en for en, i den rekkefølgen de står.

Det er noen unntak: Kontrollstrukturer (if-setninger, løkker) brukes til å velge mellom alternative setninger eller å gjenta setninger.

Ved kall på metoder i andre objekter blir kontrollen overført til det andre objektet, som utfører setningene i metoden som blir kalt. Når metoden er utført returnerer kontrollen til det opprinnelige objektet som fortsetter utførelsen rett etter metodekallet.

2.2 Kjøre java-programmer uten BlueJ

Noen synes det er en begrensning å måtte starte BlueJ for å kunne kjøre programmene sine. Det er mulig å starte programmer uten BlueJ ved å gjøre følgende:

1. Legg til en klasse som ser slik ut (Du erstatter kroppen i metoden main() med det du gjorde (menyvalgene) for å starte programmet i BlueJ. Her lager vi et nytt objekt av klassen MinKlasse, kaller det mittobjekt, og kaller metoden startMetdoe() i det nye objektet):

```
public class Main {
    public static void main(String[] args) {
        MinKlasse mittObjekt = new MinKlasse();
        mittObject.startMetode();
    }
}
```

2. Velg **`Project/Create jar file...`** i menyen i BlueJ.

3. I dialogboksen som kommer opp:

    - Velg `Main` i nedtrekksmenyen Main class.

- Include source, og BlueJ progject files er valgfritt.
- Klikk **`Continue`**
- Velg et filnavn, pass på at etternavnet (extension) blir jar, og lagre jar-filen.
- Avhengig av operativsystem kan du enten starte programmet ved å dobbeltklikke på jar-filen, eller gi kommandoen `java -jar minjarfil.jar` i et shell.

NB! NB! Det er en god grunn til at metoden main() ikke blir trukket fram i læreboka: main() er en statisk (klasse-) metode. Dvs. at den ikke kan kalle metoder i objekter uten videre. Den må lage objektene først, akkurat som du må gjøre i BlueJ.[1]

[1]Feilmeldingen du får hvis du glemmer å lage objektene er villedende, og har fått mange studenter til å sette static foran alle metodene sine. Det skal du **ikke** gjøre. Du skal lage objektene først, nøyaktig som du ville gjort i BlueJ. Så kan du kalle metoder i objektet/objektene. Det eneste stedet du skal bruke nøkkelordet static er foran main().

3 Grammatikk for Java programmer

Her finner du en noe forenklet beskrivelse av grammatikken (syntaksen) til Java. Grammatikken blir forklart ved hjelp av *maler*. Malene har navn som er skrevet i fet kursiv, f.eks: ***Klassedeklarasjon***. Malen for en klassedeklarasjon ser slik ut:

```
/** Javadoc kommentar */
public class KlasseNavn {
    Feltdeklarasjoner
    Konstruktørdeklarasjoner
    Metodedeklarasjoner
}
```

Du ser at malene ***KlasseNavn***, ***Feltdeklarasjoner***, ***Konstruktørdeklarasjoner*** og ***Metodedeklarasjoner*** blir brukt til å definere en mal for klassedeklarasjoner. Det betyr at når du skal skrive en klassedeklarasjon må du sette inn tekst som følger disse malene.

Du finner en fullstendig beskrivelse av hele programmeringsspråket Java på nettet[2], men den er ganske tung å lese.

3.1 *Klassedeklarasjon*

Et java program består av en eller flere ***Klassedeklarasjoner***. En ***Klassedeklarasjon*** ser slik ut:

```
/**
 * Beskrivelse av klassen.
 *
 * @author Forfatternavn (en tag for hver forfatter)
 * @version Dato
 */
public class KlasseNavn {
    Feltdeklarasjoner
    Konstruktørdeklarasjoner
    Metodedeklarasjoner
}
```

En klassedeklarasjone definerer en ny objekttype. Hver Klassedeklarasjonen må ha sin egen fil, som heter "***KlasseNavn***.java".

3.2 *Feltdeklarasjon*

Feltdeklarasjoner definerer variable i objektene:

```
/** Beskrivelse av variabelen. */
private Datatype variabelNavn;
```

3.3 *Konstruktørdeklarasjon*

En konstruktør er en spesiell metode som utføres rett etter at et nytt objekt er laget. Konstuktørdeklarasjoner ser slik ut:

```
/**
 * Beskrivelse av konstruktøren.
 *
 * @param parameternavn beskrivelse (en tag for hver parameter)
 */
public KlasseNavn(Parameterdeklarasjon) {
    Setninger
}
```

3.4 *Metodedeklarasjon*

Metodedeklarasjonene bestemmer hva objektene som tilhører en klasse kan gjøre.

```
/**
 * Beskrivelse av hva metoden gjør.
 *
 * @param parameternavn beskrivelse (en tag for hver parameter)
 * @return beskrivelse av hva metoden returnerer.
 */
public ReturType metodeNavn(Parameterdeklarasjon) {
    Setninger
}
```

Returtypen angir hva slag ***Datatype*** metoden returnerer. Hvis metoden ikke returnerer noen verdi brukes nøkkelordet **`void`** som returtype.

3.5 *Parameterdeklarasjon*

Parameterlisten i metodedeklarasjonen definerer de *formelle parameterne* til metoden. Deklarasjon av en parameter ser slikt ut:

```
Datatype variabelNavn
```

En metode kan ha flere parametere. Da brukes komma (,) som skilletegn mellom parameterdeklarasjonene.

3.6 *Setninger*

Setningene i en metodekropp beskriver hva som skal skje når metoden kalles. Læreboka bruker disse setningstypene. De er beskrevet i

kapittel 5.

- ***Tilordning***
- ***Metodekall***
- ***if-setning***
- ***for-løkke***
- ***for-each-løkke***
- ***while-løkke***
- ***return-setning***

3.7 *Datatype*

Navnet på datatyper brukes i deklarasjoner til å angi hva slags verdier en variabel kan inneholde og hva slags verdi en metode returnerer.

Java har to kategorier av datatyper: klasser og primitive typer. De primitive typene er **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`char`** og **`boolean`**.

3.8 Klasse-, metode- og variabelnavn

Navn i Java kan bestå av bokstaver og tall, ikke noe annet. Hvis et navn består av flere ord må det ikke være mellomrom mellom ordene.

3.8.1 *KlasseNavn*

Klassenavn bør være beskrivende for klassen.

Klassenavn skal alltid begynne med stor forbokstav. Hvis klassenavnet består av flere ord skal hvert ord ha stor forbokstav.

3.8.2 *variabelNavn*

Variabelnavnet bør beskrive hva variabelen inneholder, f.eks.: hjemmeAdresse, antallMedlemmer.

Variabelnavn skal alltid ha liten forbokstav. Hvis variabelnavnet består av flere ord skal alle ordene unntatt det første ha stor forbokstav.

3.8.3 *metodeNavn*

Metodenavnet bør beskrive hva metoden gjør, f.eks. createData(), updateDisplay(). Metodenavnet bør inneholde et verb (det representerer noe objektene kan gjøre).

Metodenavn skal alltid ha liten forbokstav. Hvis metodenavnet består av flere ord skal alle ordene unntatt det første ha stor forbokstav.

4 Data og variable

4.1 Datatyper

Data, eller verdier, i Java er delt opp i datatyper. Hver eneste mulige verdi tilhører en datatype. Java har to hovedkategorier av datatyper: objekttyper og primitive typer.

4.1.1 Primitive typer

Tabellen under viser navnet på de primitive typene, hvilke verdier som finnes i hver type, og et eksempel på en konstant.

Type	**Verdier**	**Eks.**
`byte`	-128 – 127	42
`short`	-32768 – 32767	42
`int`	-2147483648 – 2147483647	42
`long`	-9223372036854775808 – 9223372036854775807	42L
`float`	desimaltall med ca. 8 siffers nøaktighet	3.14F
`double`	desimaltall med ca. 16 siffers nøyaktighet	3.14
`char`	bokstaver og andre tegn	'a'
`boolean`	`true` og `false`	`false`

4.1.2 Objekttyper

Objekttyper er det samme som klasser. Verdiene i en objekttype er de objektene som tilhører klassen.

I motsetning til de primitive typene, som består av et begrenset antall verdier, er det ingen grense for hvor mange objekter som kan lages av hver klasse.

4.1.3 Collections og arrayer

Collections og arrayer er objekter som brukes til å holde på samlinger av objekter.

Collection-klassene er en del av standardbiblioteket. For å bruke disse klassene må du ta med en import-setning først i filen.

Når du deklarerer en variabel av en collection-type, må du si hva slags type objekter den skal inneholde. Det må du også gjøre når du lager objektet, f.eks:

```
ArrayList<Datatype> variabelNavn = new ArrayList<Datatype>();
```

Arrayer er tabeller med fast størrelse. Navnet til array-typer er navnet på innholdstypen etterfulgt av firkantparenteser. Når du lager array-objekter må du oppgi størrelsen:

```
Datatype[] variabelNavn = new Datatype[størrelse];
```

Antall plasser i arrayen angis av ***størrelse***, som er et heltallsuttrykk. Arrayobjektet består av ***størrelse*** variable av typen ***Datatype***. De heter ***variabelNavn***[***indeks***] hvor ***indeks*** er en tallverdi fra 0 til ***størrelse*** - 1.

4.2 Variable

En variabel er et sted i datamaskinens minne hvor vi kan lagre én verdi. I deklarasjonen blir variabelen gitt et navn og en type. Navnet brukes når vi skal hente eller endre verdien som er lagret i variabelen.

Typen begrenser hvilke verdier som kan lagres i variabelen. En variabel kan bare inneholde verdier som har samme type som variabelen selv. Primitive typer kan lagres direkte i variabelen. Objekter får ikke plass i variabelen. I stedet lagres adressen til objektet. Vi sier at variabelen inneholder en referanse til objektet, eller at den peker på objektet.

Det finnes tre kategorier av variable:

Felter representerer egenskaper ved objektene. Hvert objekt i en klasse har sin egen kopi av feltene som er deklarert i klassen.

Parametere er variable som brukes til å gi input data til metoder. Hvilken verdi de skal ha bestemmes ved metodekallet. Ellers er parametere som lokale variable.

Lokale variable brukes inne i metoder. Lokale variable eksisterer mens metodekallet utføres og forsvinner når det er ferdig. De brukes til å lagre mellomresultater i en metode, og kan bare brukes inne i metoden de ble deklarert i.

4.3 Uttrykk

Et uttrykk er et "regnestykke" som gir en verdi av en bestemt type som resultat.

4.3.1 *Konstanter*

Konstanter er den enkleste formelen for uttrykk. En konstant er én bestemt verdi. For eksempel er 42 en int-konstant, og "Don't␣panic!" en String-konstant.

4.3.2 *variabelNavn*

Et variabelnavn kan brukes som et uttrykk. Det gir den verdien variabelen har i øyeblikket.

Det er mulig å hente verdien til feltene i et (annet) objekt (hvis feltet er deklarert som **public**) ved å skrive:

objektNavn*.*feltNavn

hvor objektnavn er navnet på en variabel som inneholder en objektreferanse, og feltNavn er navnet på en av feltene i objektet. Ved å bruke **this** som objektNavn er det mulig å få verdien av objektets egne felter som er "skygget for" av lokale variable med samme navn.

4.3.3 *Metodekall*

Et metodekall kan brukes som uttrykk hvis metoden returnerer en verdi. Verdien av et metodekall er den verdien metoden sender fra seg i ***return-setningen***.

4.3.4 *Nytt objekt*

Vi lager nye objekter slik:

new Klassenavn(***Parameterverdier***)

Dette lager et nytt objekt av typen ***KlasseNavn***. Etter at objektet er laget blir en konstruktør kallt. Hvilken konstruktør avgjøres av parameterverdiene på samme måte som for ***Metodekall***.

Verdien av uttrykket er en referanse (peker) til det nye objektet.

4.3.5 `this`

Nøkkelordet **this** betyr "meg". Det kan brukes som en objektreferanse. Verdien er alltid en referanse til objektet selv.

4.3.6 *Sammensatte uttrykk*

Vi kan sette sammen de enkle uttrykkene over til mer kompliserte uttrykk ved å bruke operatorer. Et sammensatt uttrykk kan se slik ut:

```
Uttrykk Operator Uttrykk
```

F.eks. `x + 1`. Hvis vi har et komplisert uttrykk med mange operatorer kan vi bruke parenteser til å bestemme rekkefølgen på beregning (uttrykk inne i parenteser blir beregnet først: `2*(3+4)` gir 14 (2 ganger 7), mens `2*3+4` gir 10 (6 pluss 4).

5 *Setninger*

Setninger er den delen av koden som "gjør" noe. De kan bare brukes inne i metoder og konstruktører. De blir utført når metoden blir kallt.

5.1 *Lokal variabeldeklarasjon*

En lokal variabeldeklarasjon lager en ny variabel:

```
Datatype variabelNavn = Uttrykk;
```

Lokale variable er bare tilgjengelig mens metoden blir utført. I motsetning til parameterne til metoden får ikke en lokal variabel en verdi når metoden blir kallt, så du bør ta med ett uttrykk som gir variabelen en startverdi.

En lokal variabel kan ha samme navn som et felt i klassen. I tilfelle vil den lokale variabele "skygge for" feltet slik at bruk av variabelnavnet i andre setninger i metoden referer til den lokale variabelen.

En lokal variabel og en parameter kan ikke ha samme navn.

5.2 *Tilordningssetning*

En tilordningssetning gir en ny verdi til en variabel:

```
variabelNavn = Uttrykk;
```

Uttrykket beregnes først. Deretter blir verdien av uttrykket lagret i variabelen. Hvis variabelen blir brukt i uttrykket er det den gamle verdien som blir brukt.

5.3 *Metodekall*

Et metodekall brukes til å be et objekt om å gjøre noe. Et generelt metodekall ser slik ut:

```
objektNavn.metodeNavn(Parameterverdier);
```

objektNavn er et ***variabelNavn***. Variabelens type må være en klasse som har en metode som heter ***metodeNavn***.

For å kalle en metode i objektet som har kontrollen kan nøkkelordet **`this`** brukes som objektnavn, eller objektnavnet kan utelates helt.

Parameterverdier er en liste av ***Uttrykk*** med komma mellom. Det må være et uttrykk for hver parameter som er deklarert i metoden, og verdien av uttrykket må være av samme type som parameteren.

Litt forenklet blir metodekallet utført slik:

- Det blir opprettet en ny variabel for hver parameter. Parameterverdiene blir beregnet og lagret i disse variablene.
- Kontrollen overføres til det angitte objektet, og setningene i metodekroppen utføres der.
- Når metoden er ferdig returnerer kontrollen til objektet som kallte metoden.

5.4 *return-setning*

Return-setningen brukes til å avslutte et metodekall. Hvis metoden returnerer en verdi må du alltid ha med en return-setning som bestemmer returverdien:

```
return Uttrykk;
```

Verdien av uttrykket blir beregnet, og det er denne verdien som blir returnert fra metoden.

Hvis metoden har returtype **void** er det ikke nødvendig å bruke return fordi metoden vil returnere når siste setning er utført. Hvis du trenger den ser return-setningen slik ut:

```
return;
```

5.5 *if-setning*

En if-setning ser slik ut:

```
if (Betingelse) {
    Setninger
}
else {
    Setninger
}
```

Betingelsen beregnes først. Den er et uttrykk som gir en **boolean** verdi (**true** eller **false**). Hvis betingelsen er **true** utføres bare setningene rett etter betingelsen. Hvis betingelsen er **false** utføres bare setningene etter **else**.

Else-grenen kan utelates hvis man bare skal gjøre noe hvis betingelsen er true, og ingenting ellers:

```
if (Betingelse) {
    Setninger
}
```

Hvis man kan velge mellom flere alternativer kan man gjøre slik:

```
if (Betingelse) {
    Setninger
}
else if (Betingelse) {
    Setninger
}
else {
    Setninger
}
```

Hvis den første betingelsen er **true** utføres bare de første setningene. Hvis den første betingelsen er **false**, og den andre er **true** utføres bare de andre setningene. Hvis begge betingelsene er **false** utføres bare setningene etter siste **else**.

5.6 *while-løkke*

While-løkker brukes til å gjenta setninger så lenge en betingelse er oppfylt:

```
while (Betingelse) {
    Setninger
}
```

Betingelse er et ***Uttrykk*** som gir en **boolean** verdi. Betingelsen beregnes først. Hvis den gir verdien **true** blir setningene utført og betingelsen beregnet på nytt. Dette gjentar seg inntil beregning av betingelsen gir verdien **false**.

5.7 *for-løkke*

For-løkker brukes til å gjenta setninger når vi skal gjøre noe bestemt mellom hvert gjennomløp av løkka (f.eks. øke verdien til en variabel):

```
for (Initiering; Betingelse; Steg) {
    Setninger
}
```

Først utføres setningen ***Initiering***. Deretter beregnes betingelsen. Så lenge betingelsen gir verdien **true** utføres setningene i kroppen til løkka og setningen ***Steg*** før betingelsen beregnes på nytt.

Dette gir nøyaktig samme resultat som om vi hadde skrevet:

```
Initiering;
while (Betingelse) {
    Setninger
```

```
    Steg;
}
```

5.8 *for-each-løkke*

For-each-løkker brukes til å gå gjennom alle objektene i en Collection og gjøre noe med hvert av dem:

```
for (KlasseNavn variabelNavn : collection) {
    Setninger
}
```

collection er et uttrykk (oftest en variabel) som gir et Collection objekt som resultat. Alle objektene som er inneholdt i Collection objektet må være av typen ***KlasseNavn***.

Setningene i løkka blir gjentatt for hvert objekt i ***collection***. Før hvert gjennomløp blir ***variabelNavn*** satt til å peke på det neste objektet.

6 Javadoc

Alle klasser, metoder og felter skal ha en javadoc kommentar. Javadoc kommentarene brukes til å lage dokumentasjon for java programmer. Hensikten med å skrive javadoc er at det skal være mulig for andre å bruke klassen uten å måtte lese all koden. Derfor er det viktig å få med alt som man må kjenne til for å kunne bruke klassen. Det er nyttig for deg selv også, når det har gått noen uker og du har glemt hva du tenkte når du lagde klassen.

6.1 Bruke javadoc dokumentasjon

Du kan se dokumentasjonen for klasser du jobber med i BlueJ. Øverst til høyre i editorvinduet (der du ser koden) står det "Implementation". Trykk på den lille knappen med en trekant på, og velg "Interface" i stedet for å se dokumentasjonen.

Du finner dokumentasjon for alle standardklassene i java på cd'en som fulgte med læreboka (en zip-fil i mappen j2sdk-doc, og på nettet[4].

6.2 Javadoc kommentarer

Javadoc kommentarer begynner med `/**` og avsluttes med `*/`. Javadoc kommentaren påvirker ikke kjøringen av programmet. For å dele opp kommentaren i avsnitt bruker du en blank linje mellom hvert avsnitt slik:

```
/**
 * Hvis du skal skrive en lang kommentar er det vanlig å
 * gjøre slik som dette, med starten og slutten på kommentaren
 * på hver sin linje for seg selv.
 *
 * Dette er andre avsnitt. Stjernen (*) først i hver linje kommer
 * ikke med i dokumentasjonen.
 *
 * Tredje avsnitt.
 */
```

6.3 Javadoc tagger

De fleste javadoc kommentarer inneholder en eller flere "tagger". Taggene brukes til å merke bestemte typer informasjon.

6.3.1 @author

```
/**
 * @author Forfatternavn
 */
```

Denne taggen brukes i javadoc for klasser til å dokumentere hvem som har skrevet dem. Det skal være en @author-tag for hver forfatter. Hvis du endrer på en klasse som andre har skrevet, legger du til en ny tag med ditt navn.

6.3.2 @version

```
/**
 * @version Dato
 */
```

@version brukes i javadoc for klasser til angi hvilken versjon av klassen det er. Bruk taggen til å angi når du sist endret på klassen.

6.3.3 @param

```
/**
 * @param parameterNavn beskrivelse av parameteren.
 */
```

@param brukes til å dokumentere parametere til metoder og konstruktører. Det skal være en tag for hver parameter. Det første ordet etter taggen er parameternavnet. Resten av linjer er en beskrivelse av parameteren.

6.3.4 @return

```
/**
 * @return Beskrivelse av hva metoden returnerer.
 */
```

Denne taggen brukes i javadoc for metoder som har en annen returtype enn `void`. Den brukes til å beskrive hva metoden returnerer.

7 Klassebiblioteket

Klassebiblioteket i Java inneholder tusenvis av klasser. De mest brukte pakkene i biblioteket er java.util,java.io og java.lang. Den sistnevnte pakken java.lang inneholder mange klasser som du nesten alltid vil bruke, f.eks. String. Du trenger ikke en gang å importere den..

Pakken java.util inneholder mange generelt nyttige klasser, bl.a. Collections som brukes til å organisere mange objekter. Pakken java.io inneholder klasser som bl.a. brukes til å lese fra og skrive til filer.

For mer informasjon om klassene i standardbiblioteket se javadoc'en for biblioteket [4].

7.1 Collections

Klassene i Collection-rammeverket brukes som lagringsplassfor mange objekter. De viktigste klassene og metodene er oppsummert her.

7.1.1 ArrayList<Klasse>

ArrayList brukes når data skal lagres i rekkefølge. Når du skal bruke ArrayList må du angi hva slags objekter som kan lagres i listen. Hvis vi f.eks. skal lage en liste av studenter gjør vi slik:

```
ArrayList<Student> studenter = new ArrayList<Student>();
```

Metoden add(Klasse obj) brukes til å legge til objekter i lista.

Metoden get(int index) brukes til å hente objektet på plass nummer index.

Metoden size() returnerer lengden på lista.

7.1.2 HashMap<NøkkelType,VerdiType>

HashMap brukes når data skal finnes igjen ved å slå opp på en nøkkel. For å lage en HashMap hvor man kan finne Student-objekter ved å slå på navn (String) kan du gjøre dette:

```
HashMap<String,Student> studenter = new HashMap<String,Student>();

// legg inn en student med nøkkel "Per"
studenter.put("Per", new Student());

// hent studenten med nøkkel "Per"
Student s = studenter.get("Per");
```

7.2 I/O

Man tenker ofte på lesing og skriving av filer når det er snakk om I/O, men det omfatter også utskrifter til skjermen, sending av data over nettverk og mye annet. I standardbiblioteket for Java har man prøvd å behandle de forskjellige formene for I/O mest mulig likt. Det er f.eks. veldig liten forskjell på å skrive data til fil, og å sende data over et nettverk. I stedet er det et skille mellom binær I/O og tekst I/O. Vi bruker InputStream-objekter til å lese binære data, og OutputStream-objekter til å lese data.

7.2.1 FileInputStream

FileInputStream brukes til å lese binære data fra fil. Du lager objektet slik:

```
FileInputStream input = new FileInputStream(filNavn);
```

Variabelen filnavn må være en String som inneholder et filnavn.

Du kan bruke metoden read() til å lese en byte, og read(byte[]) til å lese et bestemt antall byte og lagre dem i arrayen.

7.2.2 FileOutputStream

FileOutputStream brukes til å skrive data til en fil. Du lager et FileOutputStream objekt slik:

```
FileOutputStream output = new FileOutputStream(filNavn);
```

Du bruker metodene write(int) til å skrive en byte, og write(byte[]) til å skrive innholdet av en byte-array til fil.

7.2.3 FileReader og FileWriter

Disse to klassene brukes på nøyaktig samme måte som FileInputStream og FileOutputStream. Den eneste forskjellen er at metodene i disse klassene opererer på tegn (den primitive typen char), og tekststrenger (String) i stedet for byte og byte[].

7.2.4 Scanner

Av en eller annen grunn er denne klassen i java.util-pakken, ikke i java.io som hadde vært mer naturlig. En Scanner brukes til å gjøre enkel parsing og tolkning av tekstinput.

Scanner har flere constructorer, som kan brukes til å lage en Scanner som leser fra f.eks. en InputStream, en Reader, eller en String, ved å gi et objekt av en av disse typene som parameter til constructoren.

Du kan bruke metodene hasNext(), hasNextBoolean(), hasNextInt(), hasNextFloat(), og enda noen til, for å sjekke om det er mer igjen å lese, og hva det er.

Det finnes tilsvarenede metoder next(), nextBoolean(), nextInt() osv., som leser inn neste symbol/tall/ord/....

8 Ordliste

Det er mange nye begreper å forholde seg til. Her er en liste over en del av dem med synonymer, og noen ord som betyr nesten det samme, synonymene først.

8.1 Engelske ord og uttrykk

abstract: abstrakt

accessor: get-metode, getter, method

application: applikasjon, anvendelse, program

argument: parameterverdi, parameter

assert: påstå

assignment: tilordning

block: blokk

body: kropp

braces: krøllparenteser

brackets: firkantparenteser

call: method call, metodekall

cohesion: sammenheng

collection: samling

compiler: kompilator

constructor: konstruktør, method

coupling: kobling

declaration: deklarasjon

encapsulate: kapsle inn, skjule

exception: unntak

expression: uttrykk

field: instance variable, property, felt, attributt, variabel

immutable object: konstant, objekt som ikke kan endres på

inherit, inheritance: arve, arv

instance: object

interface: grensesnitt

loop: løkke

map: tabell

method: metode, function, procedure

modulo: remainder, rest

mutator: set-metode, setter, method

object: objekt, instance

overload: flere metoder med samme navn

override: redefine, omdefinere, overstyre

pointer: reference, peker, referanse

run: kjøre

setter, set method: mutator, set-metode

source code: kildekode

statement: setning

state: tilstand

subclass: subklasse

superclass: superklasse

8.2 Norske ord og uttrykk

applikasjon: program, application

arv: inheritance

class: klasse

felt: field, instance variable, property, attributt, egenskap, variabel

firkantparenteser: brackets ([og])

get-metode: getter, accessor

grensesnitt: interface

konstruktør: constructor

kropp: body

krøllparenteser: braces

metode: method, funksjon, prosedyre

objekt: object, instance, eksemplar

omdefinere: redefinere, override

parameterverdi: argument, actual parametere

peker: pointer, reference

påstå, påstand: assert, assertion

redefinere: omdefinere, override

set-metode: setter, mutator

setning: statement

tabell: map (oppslagstabell)

tilordning: assignment

tilstand state

uttrykk: expression

Referanser

[1] David J. Barnes and Michael Kölling. *Objects First with Java - A practical introduction using BlueJ, 5th edition*. Pearson Education, 2012.

[2] James Gosling, Bill Joy, Guy Lewis Steele, Jr., and Gilad Bracha. *The Java Language Specification, Java SE 7 Edition*. Oracle, 2013.
`http://docs.oracle.com/javase/specs/jls/se7/html/index.html`.

[3] Oracle. Java tutorial.
`http://docs.oracle.com/javase/tutorial/index.html`.

[4] Sun. *Java API Specification*.
`http://docs.oracle.com/javase/7/docs/api/`.

Register